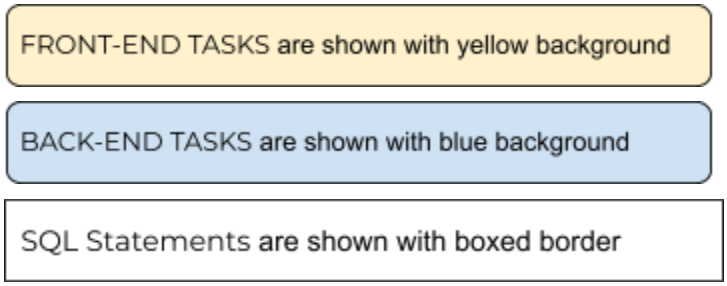
Eddy Castaneda, Paul Thi, Todd Brotze
May 12, 2021

Team 3 — Phase 2 Report

TABLE OF CONTENTS

MARKUP ANNOTATION

Example	Page
Example	Task
Example	Button or Link
Example	Input field or Form
$example	Database attribute
EXAMPLE	Database table name



DATA TYPES

USER Table

ATTRIBUTE	DATA TYPE	CONSTRAINTS
username	VARCHAR(63)	PRIMARY KEY
password	VARCHAR(63)	NOT NULL
display_name	VARCHAR(63)	NOT NULL
user_type	VARCHAR(63)	NOT NULL
email	VARCHAR(63)	
phone	VARCHAR(31)	
address	VARCHAR(63)	

RESOURCE Table

ATTRIBUTE	DATA TYPE	CONSTRAINTS
id	INTEGER	PRIMARY KEY, AUTO-INCREMENT
name	VARCHAR(63)	NOT NULL
description	VARCHAR(511)	
capabilities	VARCHAR(511)	
distance	DECIMAL(4,1)	
cost	DECIMAL(7,2)	NOT NULL
primary_function	VARCHAR(63)	NOT NULL
secondary_function	VARCHAR(63)	
unit	VARCHAR(15)	NOT NULL
user_username		FOREIGN KEY (User)

FUNCTION Table

ATTRIBUTE	DATA TYPE	CONSTRAINTS
id	INTEGER	PRIMARY KEY
function_type	VARCHAR(63)	UNIQUE, NOT NULL

UNIT Table

ATTRIBUTE	DATA TYPE	CONSTRAINTS
id	INTEGER	PRIMARY KEY
unit_type	VARCHAR(63)	UNIQUE, NOT NULL

INCIDENT Table

ATTRIBUTE	DATA TYPE	CONSTRAINTS
category_id	VARCHAR(3)	COMPOSITE PRIMARY KEY
id	INTEGER	COMPOSITE PRIMARY KEY
date	DATE	NOT NULL
description	VARCHAR(511)	NOT NULL
user_username		FOREIGN KEY (User)

CATEGORY Table

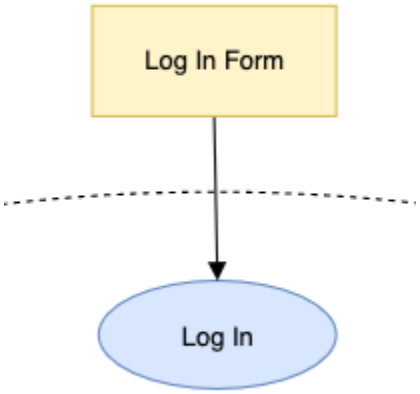
ATTRIBUTE	DATA TYPE	CONSTRAINTS
id	VARCHAR(3)	PRIMARY KEY
category_type	VARCHAR(63)	UNIQUE, NOT NULL

BUSINESS CONSTRAINTS

- Database Administrator (dBA) is solely responsible for populating the following database tables: USER, CATEGORY, FUNCTION, and UNIT. INCIDENT and RESOURCE may be populated by both the dBA and logged-in users.
- dBA sets rules for database primary key naming conventions (for example, a CATEGORY table primary key is “C1”, “C2”, “C3”, etc., versus an auto-assigned number).
- dBA or lead Administrator responsible for monitoring system logins and addressing security issues related to system access. No login encryption is provided in this scope of work.
- dBA or lead Administrator responsible for software & hardware updates/upgrades.
- dBA or lead Administrator responsible for system back-ups.
- Only a single selection is allowed for: categories, units, and primary & secondary functions fields.
- English is the primary language for this site. Any translations would need to be developed and implemented separately.
- Only the dBA is granted privileges to close and/or remove previous incidents.
- No future dates are permitted in the Incidents table.
- While duplicate-checking of resources and incidents is not in the scope of this project, database should be cleaned periodically to eliminate duplicate entries.
- This system has not been tested for old systems. Functionality may be limited on browsers or hardware.
- For any work done manually to the database, please note these rules:
    - FUNCTIONs and UNITs entered into the RESOURCE table must exist in their respective tables.
    - $category_id entered into INCIDENT table must exist in CATEGORY table
    - $id entered into must increment based on its respective $category_id INCIDENT
    - Check constraints needed on RESOURCE table to ensure primary and secondary functions are unique.
    - Check constraint that only three user types are permitted in $user_type field
    - Check constraint needed on USER table to ensure required subtype fields are NOT NULL based on user_type:
        - $user_type = “System Administrator”, then $email IS NOT NULL
        - $user_type = “CIMT User”, then $phone IS NOT NULL
        - $user_type = “Resource Provider”, then $address IS NOT NULL
- Elementary login authentication is utilized for this project, therefore the functionality of the browser’s back button is not part of this scope of work.

LOGIN

TASK DECOMPOSITION...



ABSTRACT CODE…

LOG IN FORM

- Display navigation and page elements per specification
- Validate *username* and *password* contain at least one character
- onClick **Login** button, send *username* and *password* to *Log In* task:
  - if error returned, show “Invalid Login” message at top of page

LOG IN

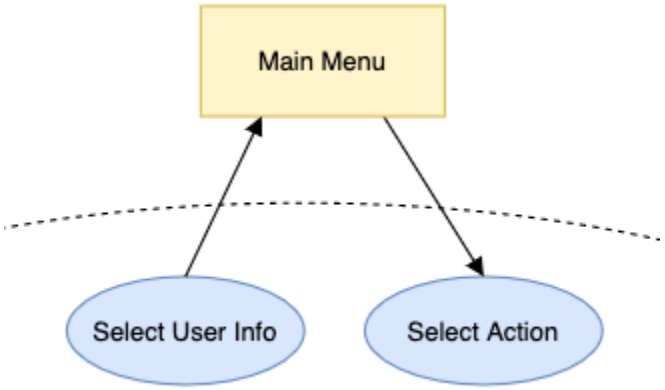
- Run query:

```
SELECT COUNT(1)=1 AS "Valid Login"
FROM user
WHERE username = username
        AND password = password;
```

- If (1) {store *username* in localStorage, then go to the **Main Menu** page}
- Else {return error message to *Log In Form* task}

MAIN MENU

TASK DECOMPOSITION...



ABSTRACT CODE…

MAIN MENU

- User successfully logged in and will remain logged in until onClick **Exit** button
- Display navigation and page elements per specification
- If (login boolean false){
  Display “You are now logged in.”
  Set login boolean to true}
- Get *username* from localStorage and send to *Select User Info*
- Save $display_name to localStorage
- If $user_type =
  - “Resource Provider”, display $display_name and $address
  - “CIMT User”, display $display_name and $phone
  - “System Administrator”, display $display_name and $email
- onClick **Add New Resource**, **Add Emergency Incident**, **Search Resources**, or **Generate Resource Report** link, run *Select Action* task
- onClick **Exit** button, clear localStorage and return to **Login** page

SELECT USER INFO

```
SELECT * FROM user
WHERE username = username;
```

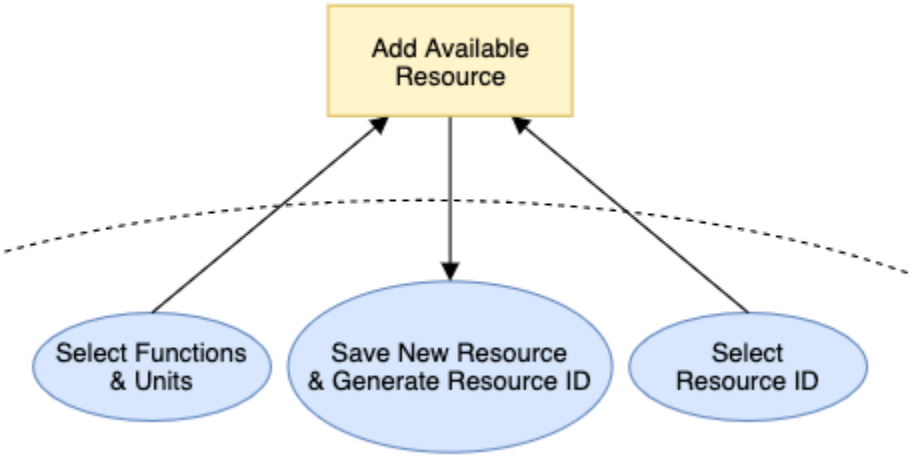
- Return results to *Main Menu* task

SELECT ACTION

- If user clicked:
  - **Add New Resource**, go to **Add New Resource** page
  - **Add Emergency Incident**, go to **Add Emergency Incident** page
  - **Search Resources**, go to **Search Resources** page
  - **Generate Resource Report**, go to **Generate Resource Report** page
  - **Exit** button, clear localStorage and return to **Login** page

ADD AVAILABLE RESOURCE

TASK DECOMPOSITION...



ABSTRACT CODE…

ADD AVAILABLE RESOURCE

- Display navigation and page elements per specification
- Call *Select Functions & Units* task, then populate:
  - *Owner* field with $display_name from localStorage
  - *Primary Function* pull-down with $function_type where $id = 1
  - *Secondary Functions* list with remaining $id(s) & $function_type(s)
  - *Unit* pull-down with $unit_type(s)
- User enters input into fields
- onChange *Primary Function*, modify *Secondary Functions* component to dynamically eliminate *Primary Function* selection
- onClick **Add** button, dynamically update *Capabilities* field to show entry and refresh input field
- If user enters *Distance from PCC*, validate entry is a number with 0.1 precision
- Validate *Cost* entry is a number up to 0.01 precision
- onClick **Cancel** button, return to **Main Menu** page

CONTINUED...

- onClick **Save** button:
  - If any required fields are empty, show error message
  - Else:
    - run *Save New Resource & Generate Resource ID* task
    - run *Select Resource ID* task, clear the form, and display results with "Resource Saved" message
- onClick **Plus Icon**, alert user they will lose existing entries:
  - If user selects **Cancel** button, remain on page
  - Else reset all fields to their original state
- onClick **Exit** button, clear localStorage and return to **Login** page

SELECT FUNCTIONS & UNITS

```
SELECT * FROM func ORDER BY id;

SELECT type FROM unit ORDER BY id;
```

- Return results to *Add Available Resource* task

SAVE NEW RESOURCE & GENERATE RESOURCE ID

```
INSERT INTO resource VALUES
    (null, name, primary_function, description, capabilities, distance, cost, unit, username);
```

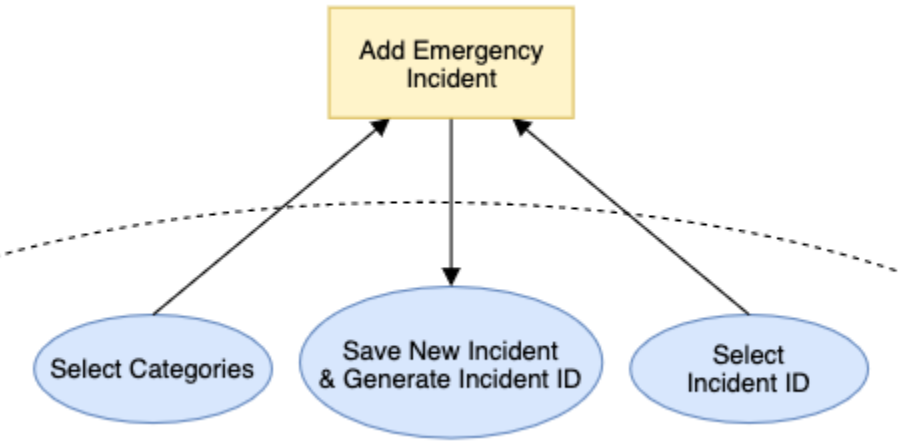
SELECT RESOURCE ID

```
SELECT * FROM resource
WHERE id = (SELECT last_insert_id()
                FROM resource
                LIMIT 1);
```

- Return results to *Add Available Resource* task

ADD EMERGENCY INCIDENT

TASK DECOMPOSITION...



ABSTRACT CODE…

ADD EMERGENCY INCIDENT

- Display navigation and page elements per specification
- User enters input into fields
- Call *Select Categories* task, then populate *Categories* drop-down with $category_name(s)
- Validate *Date* entry is “mm/dd/yy”
- onClick **Cancel** button, return to **Main Menu** page
- onClick **Save** button:
  - If any required fields are empty, show error message
  - Else:
    - run *Save New Incident & Generate Incident ID* task
    - run *Select Incident ID* task and display results with “Incident Saved” message
- onClick **Plus Icon**, alert user they will lose existing entries:
  - If user selects **Cancel** button, remain on page
  - Else reset all fields to their original state
- onClick **Exit** button, clear sessionStorage and return to **Login** page

SELECT CATEGORIES

```
SELECT type FROM category ORDER BY id;
```

- Return results to *Add Emergency Incident* task

SAVE NEW INCIDENT & GENERATE INCIDENT ID

- Generate new composite key by querying INCIDENT table

```
SELECT MAX(incident.id)
FROM incident JOIN category ON incident.category_id = category.id
WHERE category.type = category;
```

- Increment query results by 1 and insert new incident:

```
SELECT id FROM category
WHERE type = category;
```

- Take results from previous queries, as well as *username* from localStorage and insert into incident table:

```
INSERT INTO incident VALUES
  ($category_id, $id, date, description, username);
```

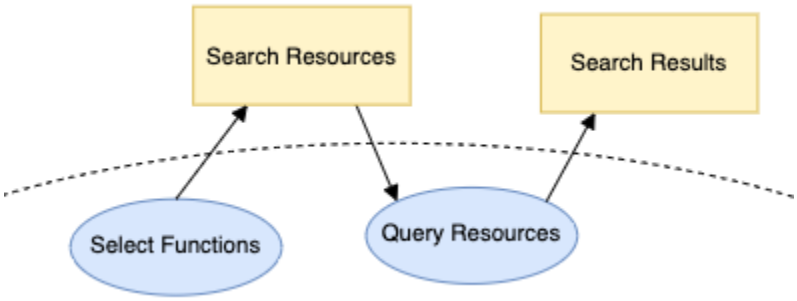
SELECT INCIDENT ID

```
SELECT * FROM incident
WHERE date = date
        description = description
        username = username;
```

- Return results to *Add Emergency Incident* task

SEARCH RESOURCES & SEARCH RESULTS

TASK DECOMPOSITION...



ABSTRACT CODE…

SEARCH RESOURCES

- Display navigation and page elements per specification
- Call *Select Functions* task, then populate *Primary Function* drop-down with $id(s) $function_type(s)
- Validate *Distance* entry is a number with 0.1 precision
- onClick **Cancel** button, return to **Main Menu** page
- onClick **Search** button, run *Query Resources* task
- onClick **Plus Icon**, alert user they will lose existing entries:
    - If user selects **Cancel** button, remain on page
    - Else reset all fields to their original state, and:
        - If *Search Results* are showing, clear *Search Results* from page
- onClick **Exit** button, clear localStorage and return to **Login** page

SEARCH RESULTS

- If *Search Results* displayed, clear *Search Results* form
- Display “Search Results” title and *Search Results* form populated with returned results, sorted by distance (closest to PCC first, null distance(s) last).

SELECT FUNCTIONS

```
SELECT * FROM func ORDER BY id;
```

- Return results to *Search Resources* task

QUERY RESOURCES

if (!*keyword* && !*primary_function* && !*distance*) {

```
SELECT * FROM resource
ORDER BY distance;
```

} else if (!*keyword* && !*primary_function* && *distance*) {

```
SELECT * FROM resource
WHERE distance < round(distance, 1)
ORDER BY distance;
```

} else if (!*keyword* && *primary_function* && !*distance*) {

```
SELECT *
FROM resource r JOIN func f ON r.primary_func_id = f.id
WHERE f.type = primary_function
ORDER BY distance;
```

} else if (*keyword* && !*primary_function* && !*distance*) {

```
SELECT * FROM resource
WHERE MATCH(name, description, capabilities)
      AGAINST('*keyword*' IN BOOLEAN MODE)
ORDER BY distance;
```

} ...

CONTINUED...

… else if (!*keyword* && *primary_function* && *distance*) {

```
SELECT *
FROM resource r JOIN func f ON r.primary_func_id = f.id
WHERE f.type = primary_function
   AND distance < round(distance, 1)
ORDER BY distance;
```

} else if (*keyword* && !*primary_function* && *distance*) {

```
SELECT * FROM resource
WHERE MATCH(name, description, capabilities)
        AGAINST('*keyword*' IN BOOLEAN MODE)
   AND distance < round(distance, 1)
ORDER BY distance;
```

} else if (*keyword* && *primary_function* && !*distance*) {

```
SELECT *
FROM resource r JOIN func f ON r.primary_func_id = f.id
WHERE f.type = primary_function
        AND MATCH(name, description, capabilities)
                AGAINST('*keyword*' IN BOOLEAN MODE)
ORDER BY distance;
```

}...

CONTINUED...

… else if (*keyword* && *primary_function* && *distance*) {

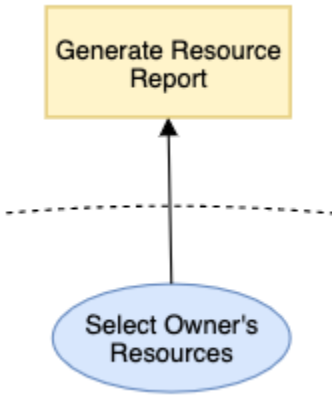
```
SELECT *
FROM resource r JOIN func f ON r.primary_func_id = f.id
WHERE MATCH(name, description, capabilities)
        AGAINST('*keyword*' IN BOOLEAN MODE)
  AND primary_func_id = primary_function
  AND distance < round(distance, 1);
```

};

- Return results to *Search Results* task

GENERATE RESOURCE REPORT

TASK DECOMPOSITION...



ABSTRACT CODE…

GENERATE RESOURCE REPORT

- Display navigation and page elements per specification
- Call *Select Owner's Resources* task
- Display returned data in *Report* form, sorted by ascending $id from the FUNCTIONS table, inclusive of all $function_type(s)
- onClick **Exit** button, clear sessionStorage and return to **Login** page

SELECT OWNER'S RESOURCES

```
SELECT func.id AS "PF ID",
                COUNT(resource.primary_func_id) AS "Count"
FROM resource JOIN func ON resource.primary_func_id = func.id
WHERE resource.user_username = "admin123"
GROUP BY func.id with rollup;

SELECT * FROM func ORDER BY id;
```

- Return results to *Generate Resource Report* task